Introduction

Data Build Tool (DBT) is a transformation framework that helps build in within workflows to our data warehouse. It compiles and runs analytics code against our data platform. There are two products which dbt offers, dbt Cloud and dbt Core. dbt Core would be the best option for us as it is an open-source tools that it is installed through the command line.

The SQL files which form the models can also contain Jinja. With this it provides a way to use control structures in queries. Also enables repeated SQL to be shared through macros.

Installation

There are a few steps involved in setting up dbt. In short it involves setting up a virtual environment, deciding on an adapter, and installing dbt and the adapter.

Install with pip

Adapters are how dbt connects with various data platforms. When installing it is important to use the correct adapter for our goals. i.e. Postgresql, Snowflake. There are two types of adapters, trusted and community. Here is a link of all the trusted adapters.

Models

Models are what most of the time is spent on within the dbt environment. They're primarily written as SELECT statements as a .sql file. DBT compiles the queries and executes them in the target database to materialise as a view or table.

Models
SQL Models

Profiles

In using DBT Core a profiles.yml file will be needed. This contains the connection details for the data platform. It facilitates the interaction between DBT and the data platform.

When DBT is run it reads the dbt_project.yml file to find the project name, then searches for a profile with the same name in the profiles.yml file.

The dbit init command is helpful in setting up a connection profile quickly.

Profiles documentation

Macros

As mentioned at the beginning, in DBT you can combine SQL with Jinja. This gives the ability to do things that aren't usually possible in SQL. They are kind of like functions in Python.

Some example of what you can do are using control structures like if and for loops, change the way the project builds based on the current target and using the results of one query to generate another query. Jinja can be used in any SQL in a DBT project, including models, analyses, tests and hooks.

Macros in Jinja are pieces of code that can be reused multiple times. Macro files can contain one or more macros.

Several useful macros have been grouped together into packages. The most popular of which is dbt-utils.

Jinja and macros documentation

Tests

Tests are assertions made about models and other resources in the DBT project. Running a test will let you know if each test in your project passes or fails. You can can test whether a specified column in a model only contains non-null values, unique values, or values that have a corresponding value in another model.

Tests can be extended to suit specific needs. Anything we can assert about a model in the form of a SELECT query can be turned into a test.

There are two ways of defining data tests:

- Singular test
- Generic data test.

If you can write a SQL query that returns failing rows, you can save that query in a .sql file in the test directory. This is a singular test.

A generic data test is a parameterised query that accepts arguments and is defined in a special test block. These tests can be referenced through yml files. DBT ships with four generic data tests built in.

Tests help to confirm that outputs and inputs are as expected.

The simplest way to define a data test is by writing the exact SQL that will return failing records. These are called singular data tests.

DBT comes with four generic data tests already defined: unique, not_null, accepted_values and relationships.

Test documentation

Seeds

Seeds are CSV files in the DBT project, typically in the seeds directory that DBT can load into the data warehouse using dbt seed command.

Seeds can be referenced in downstream models the same way as referencing models. Because the CSV files are located in the dbt repository, they are version controlled and code reviewable. Seeds are best suited to static data which changes infrequently.

Seeds are configured in the dbt_project.yml file. This means specifying settings for how seeds will be loaded and managed in the data warehouse. It might include things such as the schema, explicitly set column types or customise the delimiter.

Adding Seeds
Configuring seeds

Packages

Packages are essentially what libraries are in other coding languages. DBT packages are actually standalone projects, with models, macros and more that face specific problems. By adding packages to the project all of the resources become part of our project.

Package documentation

dbt_utils is recognised as a must have in dbt. It is also a prerequisite to other dbt packages.

dbt-expectations can be seen as an extension of generic tests. Has a wide variety of tests.

dbt-codegen is a DBT community plugin that automates creation of boilerplate code. It generates starter code for models, schema files, sources and YAML. It can help cut down on repetitive manual work.

In the case of wanting to generate unique identifiers as well as ensuring duplicate id's aren't valid the best options would be the utils and expectations packages.

Other useful resources

Community packages from dbt

Article on packages dbt-osmosis could be something to look into as a tool to make the experience better.

More packages breaks packages down into different areas.

Video short video to give an idea of what it kind of looks like.

Video A DBT setup with DBT Core for PostgreSQL.